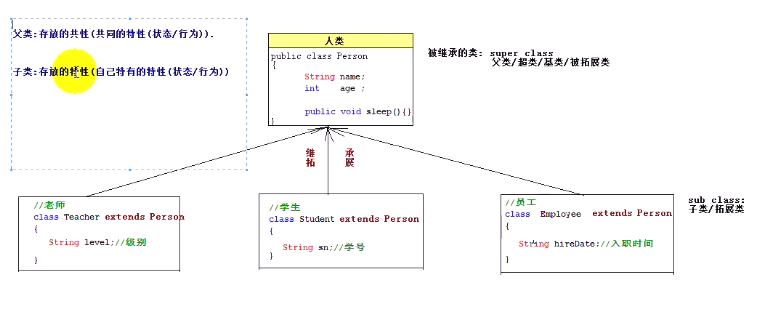
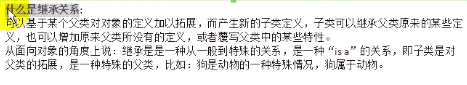
Day 08

**01引出继承关系**



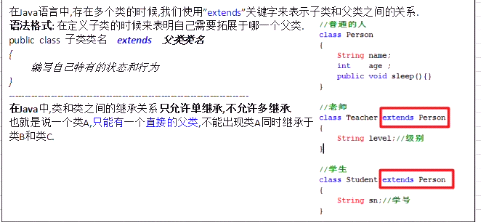
继承问题解决了代码重复（DRY原则）的问题，方便了代码的维护，但违背了封装的的思想。

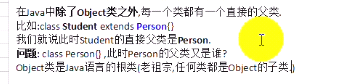
02理解继承思想



子类只能覆写父类中的方法，而不能改变属性。

从面向对象的角度来说继承是一种从一般到特殊关系。





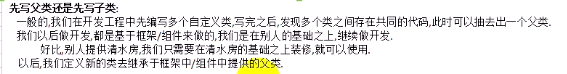
Object是java的基类，是根类

继承关系解决了代码的重复问题；真正的作用为表示出一种组合关系。

**03子类继承了父类的哪些成员**

先写父类还是先写子类：

一般在开发工程中，先编写多个自定义类，写完之后发现多个类之间存在共同的代码，可以抽取出一个父类

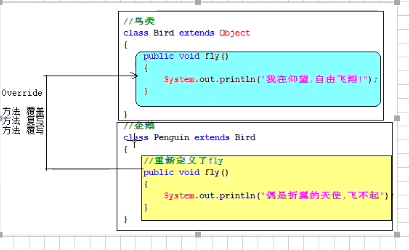


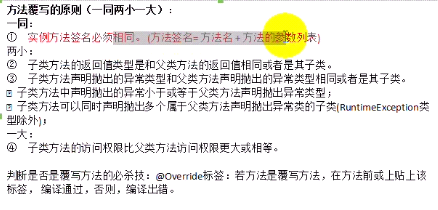
子类继承父类之后可以拥有父类的某些状态和行为，子类到底继承了父类的哪些成员根据访问控制修饰符来判断。

* Public修饰的可以被子类继承
* 缺省表示的字段和方法可以被同一个包中的子类继承
* Protected修饰的字段和方法可以被子类继承
* Private修饰的字段和方法不能被子类继承
* 父类的构造器不能被子类继承，构造器只能和当前的类名相同，但是可以通过super来引用父类的构造器

**04方法覆盖**

方法重写



两大一小的原则也符合了多态，便于多态的使用，符合里氏替换原则。

重写的方法签名必须相同；

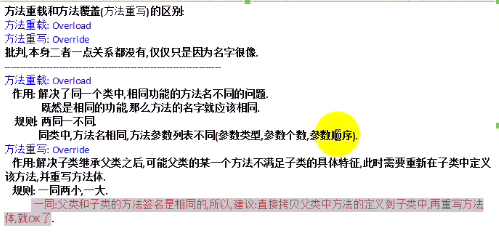
子类可以访问父类返回类型的子类型；抛出异常相同或者是其子类；

子类的访问权限比父类的访问权限至少更宽松；

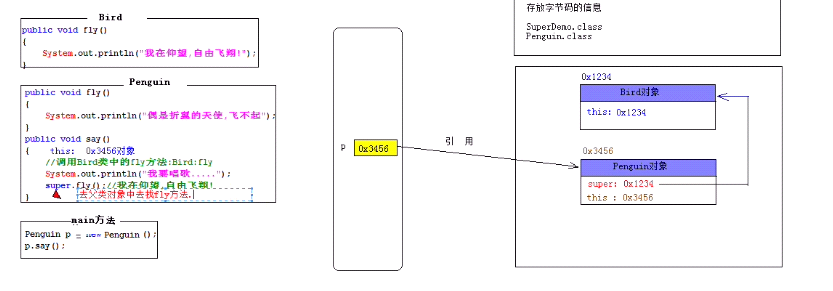
最好使用override注释来修饰覆盖的方法，以便于编译器进行编译检查

只有方法有覆盖的概念，字段不存在覆盖的概念。

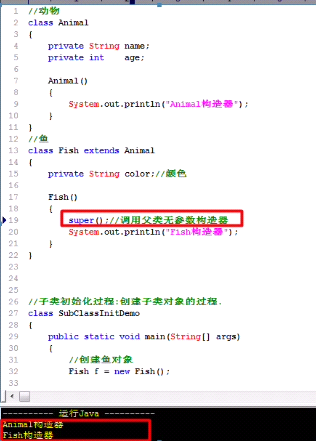
**05方法重载和方法覆盖的区别**

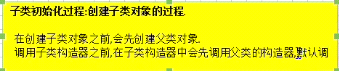


**06理解super关键字**

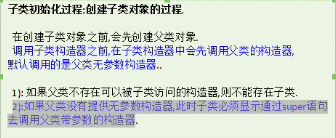


**07子类初始化过程**





因此在子类构造器中需要调用父类的构造器，如果没显示的写出调用父类的某个构造器则默认调用父类无参数构造器；如果父类不存在无参数构造器则需要在子类构造器中显示的调用有参数的构造器，否则子类对象的创建失败。



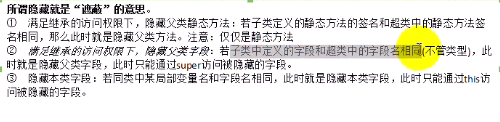
必须先有父类对象才能有子类对象。

方法调用采用就近原则：如果当前类没有方法，则到父类中寻找。

子类实例化的时候，调用子类的构造器时，该构造器会调用父类的构造器，如果没有显示的指明调用父类的构造器则默认调用父类的无参数构造器，如果父类没有无参数的构造器则编译报错；否则需要显示的指明调用父类的有参数构造器。

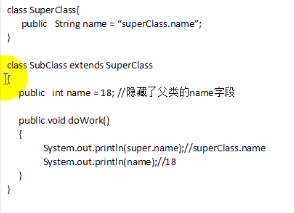
这也决定了子类实例化时会先实例化父类。

**08super引用场景和各种隐藏现象**



Super关键字的使用场景：

* 子类隐藏了父类的字段

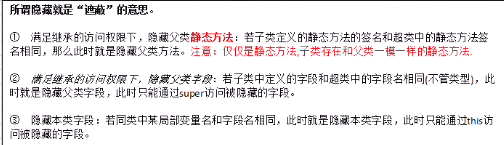


这不就是子类覆盖了父类的字段吗？

但是我们一般不讨论，因为破坏封装。

* 在子类方法中调用父类被覆盖的方法时必须使用super
* 在子类构造器中调用父类构造器，此时必须使用super语句

字段不存在覆盖，叫做隐藏

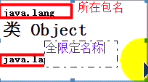


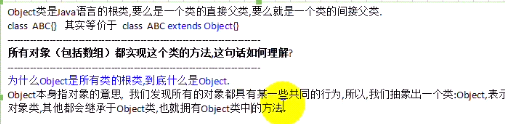
静态方法叫隐藏，非静态叫覆盖。

字段不存在覆盖，但存在隐藏。

Static不能和this以及super共存，即static修饰的方法中不能使用this和super。

**09Object类和常用方法**



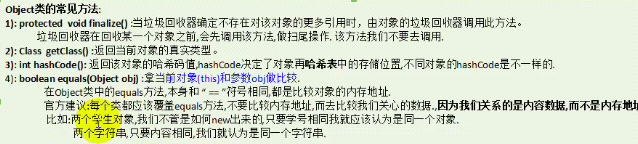


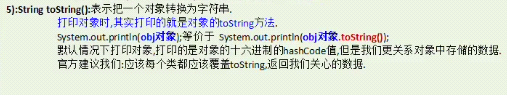
引用数据类型：类 接口 数组

在Object类中的equals方法，本身和==符号相同，都是比较对象的内存地址

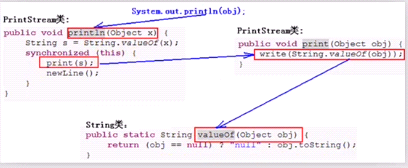
官方建议：每个类都应该覆盖equals方法，不要比较内存地址，而比较我们关心的数据。

在java中，==比较的永远是内存地址；而Object类中的equals比较的也是内存地址，建议我们覆盖该方法，以比较关键的数据。

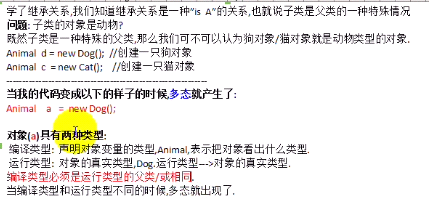




打印对象其实调用的还是对象的toString方法，底层代码实现：



**10 理解多态思想**



多态指的就是对象具有多种形态，多态的出现是因为编译类型和运行类型（运行时期是不区分类型的，实际是什么就是什么）的区分。

父类的变量或者接口变量可以引用子类的实例。

Animal a=null;

a=new Dog();

a=new Cat();

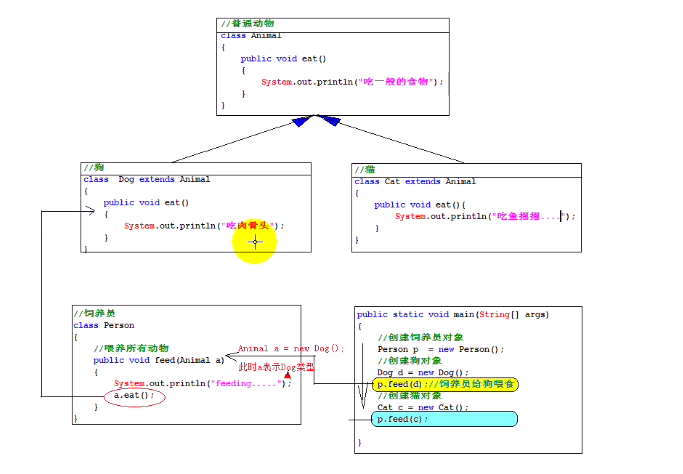
多态的前提：可以是继承关系（类和类），也可以是实现关系（接口和类），即多态不是继承关系就是实现关系。

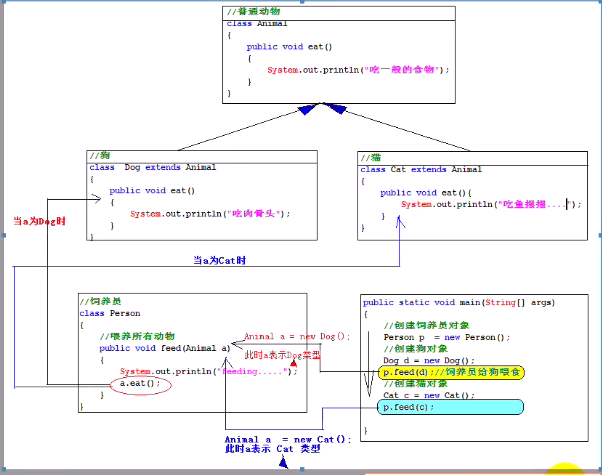
在开发中多态一般都指接口和类，也多在这方面使用。

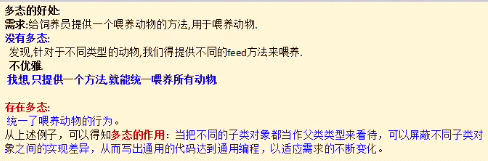


即在运行时指向子类对象的父类变量可以调用子类对象的方法，就等于是子类变量。

**11多态的好处**





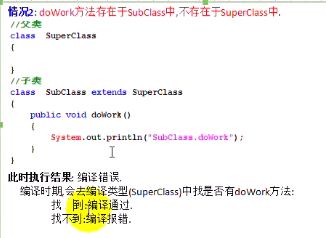


在运行时调用子类的方法

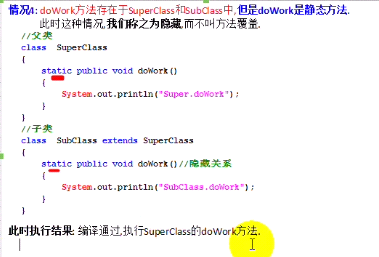
**12多态时方法调用问题**



编译时在父类中找到了方法，编译通过；**而在运行时先从子类中找方法，如果没有找到则到父类中找方法**。



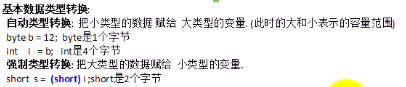
这时编译会报错，因为编译时在父类中找不到方法，这时对方法的调用不能使用多态的特征。



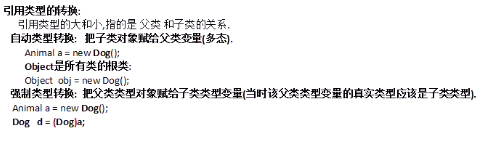
静态类型与对象无关，调用静态方法时使用的是编译类型，可以查看反编译代码。

即使是对象调用静态方法时底层实现也是由编译类型而不是对象来调用；正因为调用时是编译类型调用，所以使用多态调用静态方法的时候，在运行阶段调用的是父类的静态方法。

**13引用类型转换和instanceof运算符**



引用类型的大和小指的是父类和子类的关系：

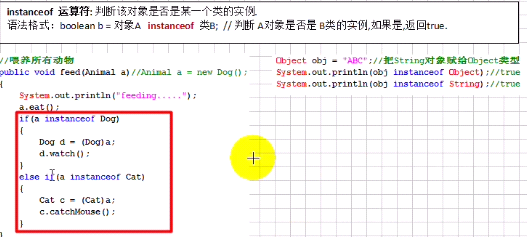


真实类型必须是子类类型才能强制转换即向下造型，才能将真实类型赋值给该类型的引用。

Animal a=new Animal();  
//Dog dog=(Dog) a; //不能将运行类型为子类型实例转换为父类型，强制转换也不可以  
Animal aa=new Dog();  
Dog dog2=(Dog)aa; //aa的运行类型或者真实是Dog所以可以强制转换

编译时 编译类型

运行时 运行类型，运行时应该是不包含类型信息的，是什么就表现出什么





对象是接口的实现类的实例或者子类的实例也会返回true。

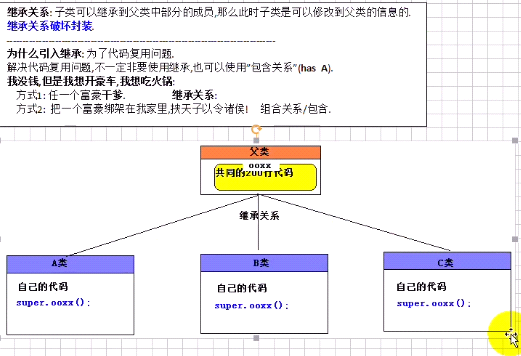
判断类型，使用反射字节码类型：

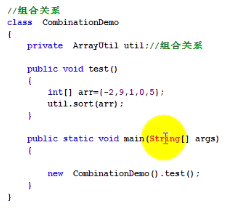
//判断准确的类型，使用字节码  
System.out.println(dog2.getClass()==Dog.class); //true  
System.out.println(dog2.getClass()==Animal.class); //error，编译不通过

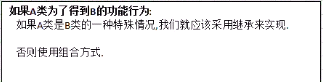


**14组合关系**

继承关系破坏封装







15小结

